Two-Factor Authentication (2FA) : Part 1

Adding an Extra Security Layer

- 2FA Application: Download any of the following
  - What is 2FA?
- SimpleTOTP class
  - SimpleTOTP.php
  - test.php
  - config.php
  - complete_demo.php

2FA Application: Download any of the following

- 📱 **Google Authenticator**
- 📱 **Microsoft Authenticator**
- 📱 **Authy**
- 📱 **1Password**

All use the same **RFC 6238 TOTP standard**

What is 2FA?

- 2FA means using two locks to keep your stuff safe!
    i. 🔑 First Lock: A secret word only you know (like your password)
    ii. 📱 Second Lock: A magic number from your phone or app

Even if someone finds your secret word,
they can’t get in without your magic number! 🛡️

How 2FA Works (from users' perspective)

```
1. User logs in with username/password ✅
2. System asks for 6-digit code 🔢
3. User opens authenticator app 📱
4. The authenticator app generates time-based code ⏰
5. User enters code → Access granted ✅
```

How 2FA Works (from an algorithm perspective)

🔑 Step 1: Generate Secret Key

Concept: Server creates shared secret

Code:

```
$secret = SimpleTOTP::generateSecret(16);
// Creates 16 random bytes for security
```

What happens:

- Server generates a random 16-byte secret
    - One time, during 2FA setup
- This secret will be shared with the user's 2FA app (like Google Authenticator) via QR code

📱 Step 2: Set up User's App

Concept: User scans QR code to add account

Code:

```
$qr_url = SimpleTOTP::getQRCodeURL($secret, $username, "My App");
// Creates: https://api.qrserver.com/v1/create-qr-code/?...
```

What happens:

- Server converts secret to Base32 format
- Creates `otpauth://` URL with secret + app info
- Generates QR code image URL
- User scans → App stores the secret

⏰ Step 3: Generate Time-Based Code

Concept: Both app and server create the same 6-digit code

Code:

```
$code = SimpleTOTP::generateCode($secret);
// Uses current time + secret → 6-digit code
```

Algorithm:

1. Get current time: `time()`
2. Create 30-second window: `floor(time() / 30)`
3. Use HMAC-SHA1: `hash_hmac('sha1', time_data, secret)`
4. Extract six digits: Dynamic truncation
5. Format: `sprintf('%06d', $code)`

✅ Step 4: Verify User Input

Concept: Server checks if the user's code matches

Code:

```
$is_valid = SimpleTOTP::verifyCode($user_input, $secret);
// Checks current time ±30 seconds for clock drift
```

Smart verification:

- Checks 3 time windows: current, +30sec, -30sec
- Handles clock differences between devices
- Uses `hash_equals()` to prevent timing attacks

🔄 Complete Flow Example

```php
// 1. Setup (one time)
$secret = SimpleTOTP::generateSecret();
$qr_url = SimpleTOTP::getQRCodeURL($secret, "john@example.com");

// 2. User scans QR code with authenticator app

// 3. Login verification (every time)
$user_enters_code = "123456"; // From their app
$is_valid = SimpleTOTP::verifyCode($user_enters_code, $secret);

if ($is_valid) {
    echo "✅ Access granted!";
} else {
    echo "❌ Invalid code!";
}
```

🎯 Key Security Features

Time-based:

- New code every 30 seconds
- Old codes automatically expire

Clock drift tolerance:

- Accepts codes from ±30 seconds
- Handles device time differences

Cryptographic security:

- HMAC-SHA1 prevents tampering
- Random secret generation
- Timing-safe comparison

SimpleTOTP class

- Simple TOTP (Time-based One-Time Password) Implementation
- Based on RFC 6238 for educational purposes

SimpleTOTP.php

generateSecret

- Generate a random secret for 2FA

```
public static function generateSecret($length = 16) {
    return random_bytes($length);
}
```

generateCode

- Generate 6-digit TOTP code from secret and timestamp

```php
public static function generateCode($secret, $timestamp = null) {
    if ($timestamp === null) { $timestamp = time(); }

    // TOTP uses 30-second time windows
    $time_slice = floor($timestamp / 30);
    // Create time-based counter (8 bytes, big-endian)
    $counter = pack('N*', 0) . pack('N*', $time_slice);
    // Generate HMAC-SHA1 hash
    $hash = hash_hmac('sha1', $counter, $secret, true);
    // Dynamic truncation (RFC 4226)
    $offset = ord($hash[19]) & 0xf;
    $code = (
        ((ord($hash[$offset+0]) & 0x7f) << 24) | ((ord($hash[$offset+1]) & 0xff) << 16) | ((ord($hash[$offset+2]) & 0xff) << 8) |
        (ord($hash[$offset+3]) & 0xff)
    ) % 1000000;

    // Return 6-digit code with leading zeros
    return sprintf('%06d', $code);
}
```

getQRCodeURL

- Generate QR code URL for authenticator apps

```php
public static function getQRCodeURL($secret, $username, $issuer = 'Demo App') {
    $secret_base32 = self::base32Encode($secret);
    $label = urlencode($issuer . ':' . $username);
    $params = http_build_query([
        'secret' => $secret_base32,
        'issuer' => $issuer,
        'algorithm' => 'SHA1',
        'digits' => 6,
        'period' => 30
    ]);

    $otpauth_url = "otpauth://totp/{$label}?{$params}";

    return "https://api.qrserver.com/v1/create-qr-code/?size=200x200&data=" . urlencode($otpauth_url);
}
```

getQRCodeURLs

- Generate multiple QR code URLs for fallback options

```php
public static function getQRCodeURLs($secret, $username, $issuer = 'Demo App') {
    $secret_base32 = self::base32Encode($secret);
    $label = urlencode($issuer . ':' . $username);
    $params = http_build_query([
        'secret' => $secret_base32,
        'issuer' => $issuer,
        'algorithm' => 'SHA1',
        'digits' => 6,
        'period' => 30
    ]);

    $otpauth_url = "otpauth://totp/{$label}?{$params}";

    return [
        'qr_server' => "https://api.qrserver.com/v1/create-qr-code/?size=200x200&data=" . urlencode($otpauth_url),
        'quickchart' => "https://quickchart.io/qr?text=" . urlencode($otpauth_url) . "&size=200",
        'otpauth_url' => $otpauth_url,
        'manual_key' => $secret_base32
    ];
}
```

base32Encode

- QR code generator uses this function to get the code.
  - `$secret_base32 = self::base32Encode($secret);`
- Convert binary secret to Base32 for QR codes

```php
public static function base32Encode($data) {
    $alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ234567'; $encoded = ''; $bits = '';

    for ($i = 0; $i < strlen($data); $i++) { $bits .= sprintf('%08b', ord($data[$i])); }
    // Pad to multiple of 5 bits
    while (strlen($bits) % 5 !== 0) {
        $bits .= '0';
    }
    // Convert every 5 bits to a base32 character
    for ($i = 0; $i < strlen($bits); $i += 5) {
        $chunk = substr($bits, $i, 5);
        $encoded .= $alphabet[bindec($chunk)];
    }
    return $encoded;
}
```

verifyCode

- Verify TOTP code against secret
- Allows 1 time window tolerance for clock drift

```
public static function verifyCode($code, $secret, $timestamp = null) {
    if ($timestamp === null) { $timestamp = time(); }
    // Check current time window and ±1 window for clock drift
    for ($i = -1; $i <= 1; $i++) {
        $test_time = $timestamp + ($i * 30);
        $expected_code = self::generateCode($secret, $test_time);
        if (hash_equals($code, $expected_code)) return true;
    }

    return false;
}
```

test.php

- Start the local server, if it is not started yet
  - `php –S localhost:8000`
- Run `curl localhost:8000/test.php`
- Purpose: Understanding the core TOTP algorithm
- What it does:
  - Standalone demonstration - Only needs SimpleTOTP.php
  - Pure algorithm focus - Shows how TOTP cryptography works
  - Step-by-step breakdown of the mathematical process
  - Educational explanations of time windows, code generation, verification

Step 1: Generate a secret

```php
echo "1. Generating Secret Key...\n";
$secret = SimpleTOTP::generateSecret(16);
$secret_base32 = SimpleTOTP::base32Encode($secret);

echo "Secret (binary): " . bin2hex($secret) . "\n";
echo "Secret (Base32): " . $secret_base32 . "\n\n";
```

```
Secret (binary): 87023e2de9658c839ec8f804e16fc3f7
Secret (Base32): Q4BD4LPJMWGIHHWI7ACOC36D64
```

Step 2: Generate current TOTP code

```php
echo "2. Generating TOTP Code...\n";
$timestamp = time();
$code = SimpleTOTP::generateCode($secret, $timestamp);

echo "Current timestamp: " . $timestamp . "\n";
echo "Time window: " . floor($timestamp / 30) . " (changes every 30 seconds)\n";
echo "Generated code: " . $code . "\n\n";
```

```
Current timestamp: 1754529861
Time window: 58484328 (changes every 30 seconds)
Generated code: 551170
```

Step 3: Verify the code

```
echo "3. Verifying TOTP Code...\n";
$is_valid = SimpleTOTP::verifyCode($code, $secret, $timestamp);
echo "Code verification: " . ($is_valid ? "✅ VALID" : "❌ INVALID") . "\n\n";
```

```
Code verification: ✅ VALID
```

Step 4: Show how codes change over time

```php
echo "4. Time-based Code Generation...\n";
for ($i = -2; $i <= 2; $i++) {
    $test_time = $timestamp + ($i * 30);  // ±60 seconds
    $test_code = SimpleTOTP::generateCode($secret, $test_time);
    $time_label = $i === 0 ? "CURRENT" : ($i > 0 ? "+{$i}×30s" : "{$i}×30s");

    echo sprintf("Time %s: %s → Code: %s\n",
        $time_label, date('H:i:s', $test_time),  $test_code
    );
}
```

```
Time -2×30s: 01:23:21 → Code: 381904
Time -1×30s: 01:23:51 → Code: 686102
Time CURRENT: 01:24:21 → Code: 551170
Time +1×30s: 01:24:51 → Code: 992161
Time +2×30s: 01:25:21 → Code: 722375
```

Step 5: QR Code URL

```
echo "5. QR Code for Authenticator App...\n";
$qr_url = SimpleTOTP::getQRCodeURL($secret, 'testuser', 'Demo App');
echo "QR Code URL: " . $qr_url . "\n\n";
```

```
QR Code URL: https://api.qrserver.com/v1/create-qr-code/?size=200x200&data=otpauth
%3A%2F%2Ftotp%2FDemo%2BApp%253Atestuser%3Fsecret
%3DQ4BD4LPJMWGIHHWI7ACOC36D64%26issuer%3DDemo%2BApp%26algorithm
%3DSHA1%26digits%3D6%26period%3D30
```

Step 6: Test invalid codes

```php
echo "6. Testing Invalid Codes...\n";
$invalid_codes = ['000000', '123456', '999999'];

foreach ($invalid_codes as $invalid_code) {
    $is_valid = SimpleTOTP::verifyCode($invalid_code, $secret, $timestamp);
    echo "Code {$invalid_code}: " . ($is_valid ? "✅ VALID" : "❌ INVALID") . "\n";
}
```

```
Code 000000: ❌ INVALID
Code 123456: ❌ INVALID
Code 999999: ❌ INVALID
```

Step 7: Clock drift tolerance

```php
$old_code = SimpleTOTP::generateCode($secret, $timestamp - 30);  // Previous window
$future_code = SimpleTOTP::generateCode($secret, $timestamp + 30);  // Next window

echo "Previous window code: {$old_code} → " .
     (SimpleTOTP::verifyCode($old_code, $secret, $timestamp) ? "✅ ACCEPTED" : "❌ REJECTED") . "\n";
echo "Current window code:  {$code} → " .
     (SimpleTOTP::verifyCode($code, $secret, $timestamp) ? "✅ ACCEPTED" : "❌ REJECTED") . "\n";
echo "Future window code:   {$future_code} → " .
     (SimpleTOTP::verifyCode($future_code, $secret, $timestamp) ? "✅ ACCEPTED" : "❌ REJECTED") . "\n";
```

```
Previous window code: 686102 → ✅ ACCEPTED
Current window code:  551170 → ✅ ACCEPTED
Future window code:   992161 → ✅ ACCEPTED
```

config.php

- This script contains utility functions for storing and loading users in JSON.

```php
<?php
// Session file to persist user data (simple file-based storage)
$users_file = 'users_data.json';

// Load users from file if it exists
function loadUsers() {
    global $users, $users_file;
    if (file_exists($users_file)) {
        $data = json_decode(file_get_contents($users_file), true);
        if ($data) { $users = $data; }
    }
}
// Save users to file
function saveUsers() {
    global $users, $users_file;
    file_put_contents($users_file, json_encode($users, JSON_PRETTY_PRINT));
}
```

- It contains helper functions to get users and update users.

```php
// Helper function to get user by username
function getUser($username) {
    global $users;
    loadUsers();
    return isset($users[$username]) ? $users[$username] : null;
}

// Helper function to update user
function updateUser($username, $data) {
    global $users;
    loadUsers();
    if (isset($users[$username])) {
        $users[$username] = array_merge($users[$username], $data);
        saveUsers();
        return true;
    }
    return false;
}

// Initialize users file
loadUsers();
?>
```

complete_demo.php

- Run `curl localhost:8000/demo_complete.php`
- Purpose: Complete 2FA system in a production-like environment
- What it does:
  - Full system simulation - Uses config.php, user database
  - Complete user workflow - Setup → Verification → Login → Reset
  - Real-world integration - Shows how 2FA fits into actual applications
  - System management - User state, data persistence, error handling

users_data.json

- We use JSON for storing user/password information.
  - We will use MySQL later in this course.
  - Each user may use TOTP for the user name/password.

```
{
    "john": {
        "id": 1,
        "username": "john",
        "password": "password123",
        "totp_secret": null,
        "totp_enabled": false
    },
    "admin": {
        "id": 2,
        "username": "admin",
        "password": "admin123",
        "totp_secret": "xycIprnjz52rELZwKzPdYA==",
        "totp_enabled": true
    }
}
```

Step 1: Test QR Code Generation

```php
$secret = SimpleTOTP::generateSecret(16);
$secret_hex = bin2hex($secret);
$secret_base32 = SimpleTOTP::base32Encode($secret);

echo "✅ Generated secret: {$secret_hex}\n";
echo "✅ Base32 format: {$secret_base32}\n";

// Test old vs new QR URLs
$qr_urls = SimpleTOTP::getQRCodeURLs($secret, $demo_user, 'PHP 2FA Demo');
echo "✅ QR Server URL: " . substr($qr_urls['qr_server'], 0, 60) . "...\n";
echo "✅ QuickChart URL: " . substr($qr_urls['quickchart'], 0, 60) . "...\n";

// Generate a TOTP code for demonstration
$current_code = SimpleTOTP::generateCode($secret);
echo "✅ Current TOTP code: {$current_code}\n\n";
```

```
✅ Generated secret: 8c97456e5ef0d76e9c9cddec2030e7d3
✅ Base32 format: RSLUK3S66DLW5HE43XWCAMHH2M
✅ QR Server URL: https://api.qrserver.com/v1/create-qr-code/?size=200x200&dat...
✅ QuickChart URL: https://quickchart.io/qr?text=otpauth%3A%2F%2Ftotp%2FPHP%2B2...
✅ Current TOTP code: 897279
```

Step 2: Check Initial User Status

```php
$user = getUser($demo_user);
if (!$user) {
    echo "❌ User not found. Creating demo environment...\n";
    // This would typically be handled by config.php initialization
} else {
    echo "✅ User found: {$demo_user}\n";
    echo "   TOTP Enabled: " . ($user['totp_enabled'] ? 'Yes' : 'No') . "\n";
    echo "   Has Secret: " . (!empty($user['totp_secret']) ? 'Yes' : 'No') . "\n";
}
echo "\n";
```

```
✅ User found: john
   TOTP Enabled: No
   Has Secret: No
```

Step 3: Reset if already enabled (clean slate)

- For test purposes, the users should *not* be disabled from using 2FA.

```
if ($user && $user['totp_enabled']) {
    echo "ℹ️  2FA already enabled. Resetting for demo...\n";
    updateUser($demo_user, [
        'totp_secret' => null,
        'totp_enabled' => false
    ]);
    echo "✅ Reset completed\n";
} else {
    echo "✅ User ready for 2FA setup\n";
}
echo "\n";
```

```
✅ User ready for 2FA setup
```

Step 4: Simulate 2FA Setup

- In this mode, users cannot use the QR code, so we simulate users' input to verify.

```php
try {
    // Generate new secret for setup
    $setup_secret = SimpleTOTP::generateSecret();

    // Store secret (temporarily, not yet enabled)
    updateUser($demo_user, ['totp_secret' => base64_encode($setup_secret)]);

    // Generate QR code information
    $setup_qr_urls = SimpleTOTP::getQRCodeURLs($setup_secret, $demo_user, 'PHP 2FA Demo');
    $setup_code = SimpleTOTP::generateCode($setup_secret);

    echo "✅ Secret generated and stored\n";
    echo "✅ QR code URL ready: " . substr($setup_qr_urls['qr_server'], 0, 50) . "...\n";
    echo "✅ Manual entry key: " . SimpleTOTP::base32Encode($setup_secret) . "\n";
    echo "✅ Current code for verification: {$setup_code}\n";

} catch (Exception $e) {
    echo "❌ Setup failed: " . $e->getMessage() . "\n";
    exit(1);
}
echo "\n";
```

```
✅ Secret generated and stored
✅ QR code URL ready: https://api.qrserver.com/v1/create-qr-code/?size=2...
✅ Manual entry key: WK7KCTGMIUH3RLWM3LP3JFOTNA
✅ Current code for verification: 515736
```

Step 5: Simulate Verification

- We enable 2FA for the demo user.

```
// Reload user data
$user = getUser($demo_user);
if ($user && !empty($user['totp_secret'])) {
    $stored_secret = base64_decode($user['totp_secret']);
    // Verify the code we generated
    $is_valid = SimpleTOTP::verifyCode($setup_code, $stored_secret);
    if ($is_valid) {
        // Enable 2FA
        updateUser($demo_user, ['totp_enabled' => true]);
        echo "✅ Code verification successful\n";
        echo "✅ 2FA enabled for user: {$demo_user}\n";
    } else {
        echo "❌ Code verification failed\n";
    }
} else { echo "❌ No secret found for verification\n"; }
```

```
✅ Code verification successful
✅ 2FA enabled for user: john
```

Step 6: Test Login

- In the verification, we assume that users give the correct $setup_code.

```php
$user = getUser($demo_user);
if ($user && $user['totp_enabled']) {
    $login_secret = base64_decode($user['totp_secret']);
    $login_code = SimpleTOTP::generateCode($login_secret);

    // Simulate login verification
    $login_valid = SimpleTOTP::verifyCode($login_code, $login_secret);

    echo "✅ Generated login code: {$login_code}\n";
    echo "✅ Login verification: " . ($login_valid ? 'SUCCESS' : 'FAILED') . "\n";
} else {
    echo "❌ 2FA not properly enabled\n";
}
echo "\n";
```

```
✅ Generated login code: 515736
✅ Login verification: SUCCESS
```

Step 7: Demonstrate Time Window Behavior

```
if ($user && $user['totp_enabled']) {
    $time_secret = base64_decode($user['totp_secret']);
    $current_time = time();

    echo "Current time: " . date('H:i:s', $current_time) . " (timestamp: {$current_time})\n";
    echo "Time window: " . floor($current_time / 30) . "\n\n";

    echo "Code generation for different time windows:\n";
    for ($i = -2; $i <= 2; $i++) {
        $test_time = $current_time + ($i * 30);
        $test_code = SimpleTOTP::generateCode($time_secret, $test_time);
        $time_label = $i === 0 ? "CURRENT" : ($i > 0 ? "+{$i}×30s" : "{$i}×30s");
        $is_accepted = SimpleTOTP::verifyCode($test_code, $time_secret, $current_time);

        echo sprintf("  %s: %s → %s %s\n",
            str_pad($time_label, 8),
            date('H:i:s', $test_time),
            $test_code,
            $is_accepted ? '✅ ACCEPTED' : '❌ REJECTED'
        );
    }
}
```

```
Current time: 02:03:21 (timestamp: 1754532201)
Time window: 58484406

Code generation for different time windows:
  -2×30s : 02:02:21 → 769126 ❌ REJECTED
  -1×30s : 02:02:51 → 249631 ✅ ACCEPTED
  CURRENT : 02:03:21 → 515736 ✅ ACCEPTED
  +1×30s : 02:03:51 → 845397 ✅ ACCEPTED
  +2×30s : 02:04:21 → 671533 ❌ REJECTED
```

Step 8: Demonstrate Reset Functionality

```php
echo "Current user state before reset:\n";
$user = getUser($demo_user);
echo "  - TOTP Enabled: " . ($user['totp_enabled'] ? 'Yes' : 'No') . "\n";
echo "  - Has Secret: " . (!empty($user['totp_secret']) ? 'Yes' : 'No') . "\n";

// Perform reset
$reset_data = [
    'totp_secret' => null,
    'totp_enabled' => false
];

$reset_success = updateUser($demo_user, $reset_data);

if ($reset_success) {
    echo "✅ Reset successful\n";

    // Verify reset
    $user = getUser($demo_user);
    echo "User state after reset:\n";
    echo "  - TOTP Enabled: " . ($user['totp_enabled'] ? 'Yes' : 'No') . "\n";
    echo "  - Has Secret: " . (!empty($user['totp_secret']) ? 'Yes' : 'No') . "\n";
} else {
    echo "❌ Reset failed\n";
}
```

```
Current user state before reset:
  - TOTP Enabled: Yes
  - Has Secret: Yes
✅ Reset successful
```